CENTRO UNIVERSITÁRIO DE BRASÍLIA - UniCEUB
Faculdades de Ciências Exatas - FAET
Curso de Engenharia da Computação

AUTOMAÇÃO RESIDENCIAL POR MEIO DO CELULAR

por

BRUNO APRIGIO E SILVA

Trabalho submetido à avaliação,

como requisito parcial para a obtenção do Certificado de conclusão do curso

de Engenharia da Computação

Prof. M. Sc. Antonio José Gonçalves Pinto
Orientador

Prof. Francisco Javier
Professor da Disciplina

Brasília, junho de 2006.

BRUNO APRIGIO E SILVA

AUTOMAÇÃO RESIDENCIAL POR MEIO DO CELULAR

COMISSÃO EXAMINADORA

_____

Prof. Orientador

_____

[Nome do Examinador]

_____

[Nome do Examinador]

Brasília, 19 de julho de 2006.

Dedico este trabalho aos meus pais Fernando e Ane, minha noiva Amanda e minha irmã Renata que por serem pessoas inigualáveis me cobrem de estímulos e me impulsionam a nunca desistir dos meus sonhos e objetivos. Dispenso aqui meus agradecimentos por terem aceitado se privar de minha companhia pelos estudos, concedendo a mim a oportunidade de vencer mais essa barreira.

Agradeço primeiramente ao meu orientador Professor Antônio Gonçalves a fim de retribuir a competência e a disponibilidade com que sempre me orientou, contribuindo para um maior aprimoramento profissional e pessoal que acredito ter adquirido por meio do vínculo estabelecido.

Agradeço ainda, aos meus companheiros no âmbito profissional e aos meus amigos de faculdade, sempre disponíveis nos momentos de dificuldade.

RESUMO

Neste trabalho é apresentado a implementação de uma interface capaz de promover a interação com o usuário, fazendo com que esta envie requisições por meio de uma conexão utilizando o protocolo HTTP. As requisições serão recebidas por uma aplicação servidora composta de um servidor web e de um software utilizando a linguagem JAVA. A aplicação servidora efetuará a interpretação da requisição e encaminhará para a porta paralela que se comunicará com um hardware que simulará todo o processo de automação.

Palavras-chave: Automação Residencial, Java, J2ME, HTTP, Conector DB25.

ABSTRACT

In this document it is present the implementation of an interface capable to promote the interaction with the user, making with that this sends solicitations by means of a connection using protocol HTTP. The solicitations will be received by a composed serving application from a server web and a software using language JAVA. The serving application will effect the interpretation of the solicitation and will direct for the parallel door that will be communicated with the hardware that will simulate the automation process all.

Keywords:Residential Automation, JAVA, J2ME, HTTP, Connector DB25.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

CAPÍTULO 1

INTRODUÇÃO

Em 1983, nos Estados Unidos, surge o primeiro celular aprovado pelo FCC (*Federal Communication Commission*) o DinaTAC 800X da Motorola dando início ao uso comercial da tecnologia móvel no país e no mundo.

A tecnologia móvel revolucionou o mercado com a idéia de manter o usuário sempre conectado, lembra Ruddy Krolopp, um dos primeiros participantes da equipe de desenvolvimento do aparelho: "*Consumidores ficaram tão impressionados com a idéia de estar sempre conectados que se dispunha a pagar US$ 3,995*".

Passados 20 anos do início comercial da telefonia móvel, o conceito em torno do uso dos dispositivos móveis mudou, os consumidores buscam reforçar seus estilos de vida.

Com o barateamento da tecnologia, o número de usuários de celular no mundo passou de cerca de 300 mil, em 1984, para mais de 1,2 bilhão, atualmente. À medida que a indústria cresceu, as empresas anteciparam a demanda por tecnologias inovadoras, segundo artigo da PUC Rio.[PUC 02]

1.1 MERCADO BRASILEIRO

Há oito anos, quando foi iniciado o processo de privatização do Sistema Telebrás, os profissionais da área, assim como o mercado, possuíam convicção que essa seria a medida certa a ser adotada. Tal certeza deve-se a falta de competição, onde o Estado atuava como

órgão atuador e regulador, a baixa capacidade de investimentos e vários outros motivos, os quais somente poderiam ser solucionados através da privatização.

Em julho de 1995, uma emenda constitucional abriu caminho para as privatizações, ocorridas nos anos de 1997 e 1998. No dia 07 de Outubro de 1997, o projeto de lei que regulamenta a ANATEL (*Agência Nacional de Telecomunicações*) é aprovado. Surge, então, o órgão regulador das telecomunicações no Brasil, com a missão de estabelecer a competição a partir de limites pré-estabelecidos. Em meio a tudo isso, houve a explosão da telefonia móvel, com impressionante crescimento no número de acessos e na quantidade aparelhos em poder dos usuários.[MOR 06]

Deste então, o mercado de desenvolvimento de aplicações para celulares utilizando a tecnologia J2ME vem crescendo a passos largos. Com o freqüente lançamento de aparelhos cada vez mais modernos e que disponibilizam cada vez mais serviços e recursos para o usuário, o crescimento se torna algo natural.

Segundo notícia do Jornal Gazeta Mercantil em matéria publicada no dia 12 de Janeiro de 2005, a adesão das pessoas ao uso de telefones celulares vem crescendo em relação ao mercado de telefonia fixa no Brasil. Os números apresentados no quadro a seguir demonstram uma estabilização no mercado de telefonia fixa com relação ao número de clientes e um potencial mercado da telefonia móvel. Atualmente o número de celulares móveis já ultrapassa o número de clientes da telefonia fixa, como mostrado no quadro da Figura 1.1.

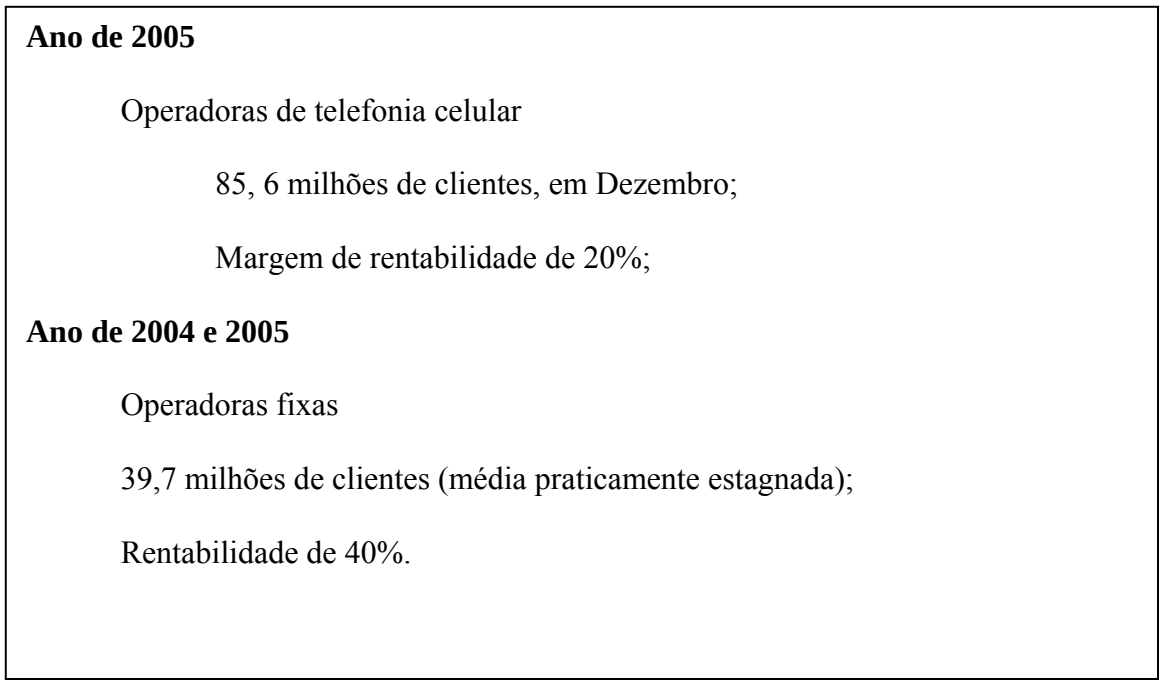

FIGURA 1.1 – Comparativo do crescimento entre a telefonia móvel e fixa.

A facilidade oferecida por essa tecnologia impulsiona um mercado fabuloso em relação à movimentação de dinheiro, seja através de receitas para as operadoras, seja através de negócios no mundo das telecomunicações.

1.2 OBJETIVO

É fato que a tecnologia celular surgiu para mudar os conceitos no campo das telecomunicações. E é nessa temática que surge a idéia de explorar os recursos disponibilizados a fim de facilitar a vida dos usuários. Logo, retornando juntamente com a tecnologia JAVA ao seu propósito, ou seja, a criação de programas para os dispositivos eletrônicos inteligentes destinados ao consumidor final, surge à possibilidade da criação de um serviço que por meio de um simples toque dá ao usuário o controle de sua residência.

Desta forma, o objetivo deste é demonstrar a viabilidade em disponibilizar um serviço que integrado com a tecnologia de automação residencial possibilite ao usuário obter o acesso

remoto aos dispositivos integrantes de sua residência. Criando desta forma um serviço diferenciado, e a possibilidade de explorar dois mercados emergentes que são os das casas inteligentes (por meio da automação residencial) e o mercado de telefonia móvel.

O trabalho está organizado da seguinte forma:

- Capítulo 1 - Faz a introdução do trabalho, apresentando o tema do projeto, a motivação e os objetivos da pesquisa;
- Capítulo 2 - Aborda a base de conhecimento teórico necessária para o entendimento e desenvolvimento do projeto;
- Capítulo 3 - Descreve todo o projeto baseado na disponibilização de um serviço seguindo a proposta de automação residencial (casas inteligentes). Neste serão apresentados detalhes de implementação e explicações inerentes à arquitetura do projeto;
- Capítulo 4 - Apresenta os resultados obtidos traçando um paralelo com a possibilidade da criação de um projeto competitivo para o mercado;
- Capítulo 5 - Traz as conclusões do trabalho e as sugestões de estudos futuros.

CAPÍTULO 2

FERRAMENTAS TECNOLOGICAS

Este capítulo faz referência às ferramentas e tecnologias que proporcionaram o embasamento teórico para o desenvolvimento do projeto. As ferramentas e tecnologias apresentadas se caracterizam por serem integrantes do projeto *open source* e por serem produtos de mercado.

2.1 AUTOMAÇÃO RESIDENCIAL

A automação residencial consiste em novas tecnologias que procuram oferecer conforto, praticidade, produtividade, economia, eficiência, e rentabilidade, com a valorização da imagem do empreendimento e de seus usuários.

Segundo Roberto Luigi Bettoni, diretor da Bettoni Automação e Segurança Ltda, em artigo publicado no site portal da automação "Primeiramente foi a automação industrial, ligada ao controle e à supervisão das linhas de produção, depois a de edifícios comerciais, mais voltadas às áreas patrimoniais e institucionais. Finalmente, chegamos a automação residencial, um mercado que já é realidade em todo o Brasil, com soluções interessantes e diferenciadas voltadas aos serviços para o usuário".

Ainda segundo Bettoni, o que se objetiva com a automação residencial é a integração de tecnologias de acesso à informação e entretenimento, com otimização dos negócios, da Internet, da segurança, além de total integração da rede de dados, voz, imagem e multimídia. Isso é obtido através de um projeto único que envolve infra-estrutura, dispositivos e software

de controle, buscando garantir ao usuário a possibilidade de controle e de acesso a sua residência a distancia, dentro ou fora da mesma.

Assim sendo, torna-se viável comandar grande parte das instalações da unidade através de um controle remoto, um aparelho celular e via Internet, a qual se tornou mais fácil o acesso a novos serviços de comunicação.

Em relação ao controle via Internet ou por meio do aparelho celular, a sofisticação e as possibilidades são inúmeras. O usuário poderá, por exemplo, em seu escritório necessitando apenas estar conectado a uma rede, em qualquer lugar do mundo, receber alarmes de intrusão, mau funcionamento de aparelhos, programar tarefas de ligar ou desligar eletrodomésticos, ascender ou apagar luzes, dentre muitas outras possibilidades. [BET 03]

Sendo assim, a fim de oferecer um serviço baseado em uma automação residencial se torna necessário a apresentação de algumas tecnologias que serão utilizadas. Logo, a seguir, será apresentada toda a tecnologia necessária para desenvolver um projeto que integre a tecnologia de automação residencial e telefonia móvel.

2.2 TECNOLOGIA JAVA

A tecnologia Java é basicamente uma linguagem de programação de alto nível desenvolvida pela *Sun Microsystems* originada do C++. Uma característica marcante é a sua independência em relação à plataforma: a tecnologia Java pode rodar com o mesmo código sobre as plataformas Windows, Linux, Solaris, Machintosh, dentre outros. Por meio de uma *JVM (Java Virtual Machine)* o código escrito apenas é interpretado. Outra característica é que a Java possui uma sintaxe similar a do C++, o que torna mais fácil o aprendizado para os programadores conhecedores dessa linguagem. Java é uma linguagem completamente orientada a objetos. Tudo em Java, exceto poucos tipos primitivos como os números, são

objetos. Foi difundida após o estouro da *World Wide Web* (Internet) e é utilizada em larga escala para o desenvolvimento de aplicativos para a mesma. [WIK 06] e [DEI 01]

2.2.1 Principais Características

A seguir estão identificadas as principais características da tecnologia Java: simples, orientada a objetos, familiar, distribuída, robusta, segura, possui arquitetura neutra, portável, interpretada, alta performance, *multithread* e dinâmica. [WIK 06] e [DEI 01] e [PAM 06]

Simples, Orientada a Objetos e Familiar

A primeira característica da Java é possuir uma linguagem simples que pode ser programada sem um extensivo programa de treinamento. A concepção fundamental da tecnologia Java é que ela possa ser rapidamente absorvida. Possibilitando que os programadores sejam produtivos em um curto espaço de tempo. A orientação a objetos é uma técnica focada nos objetos e interfaces para aqueles objetos. A técnica faz analogia aos objetos reais. Ou seja, um automóvel possui características e atributos específicos. Ainda possui a sua familiaridade com C++.

Robustez e Segurança

A Java foi desenvolvida para criar softwares de grande confiança. O modelo de manipulação de memória é extremamente simples: objetos são criados com o operador **new**. Não é necessário que o programador manipule os ponteiros ou a memória. Tal fato elimina classes com a função de tratar os erros como as existentes nos programas em C e C++.

A tecnologia Java foi desenvolvida para operar em ambientes de sistemas distribuídos, o que significa que a segurança é algo importante. Nos ambientes de trabalho, aplicações escritas em Java são seguras contra intrusões não autorizadas e vírus desenvolvidos para a invasão de sistemas.

Arquitetura Neutra e Portabilidade

A tecnologia Java possui uma arquitetura desenvolvida para ser utilizada em ambientes heterogêneos. Para atender a essa necessidade de sistemas operacionais o *Compilador Java* produz os *bytecodes* – uma arquitetura intermediária neutra e eficiente para que pode ser utilizada por vários *hardwares* e plataformas.

A arquitetura neutra é apenas uma parte da portabilidade da tecnologia Java. A portabilidade do Java é conhecida pela *Java Virtual Machine*. É uma especificação que uma máquina abstrata que juntamente com o compilador Java pode gerar códigos nativos.

Alta Performance

A Java consegue um desempenho superior porque pode traduzir os *bytecodes* em tempo de execução enquanto outras aplicações rodam ao mesmo tempo.

Interpretada, *Multithread,* e Dinâmica

O Interpretador Java pode executar os *bytecodes* diretamente em qualquer máquina que possua suporte. Sendo que este é um processo leve, possibilitando que o processo de desenvolvimento seja mais rápido e exploratório.

A Java possui os benefícios do *Multithreading* responsáveis por uma melhor iteratividade e comportamento em tempo real. Ainda foi desenvolvida para adaptar-se a vários ambientes. Bibliotecas podem livremente adicionar novos métodos e instancias de variáveis sem nenhum efeito no cliente.

2.2.2 A História da Tecnologia Java

Em 1991, na empresa *Sun Microsystems*, foi iniciado o *Green Project* que posteriormente daria origem ao Java. Os membros do projeto eram Patrick Naughton, Mike Sheridan e James Gosling. O projeto tinha como objetivo se antecipar ao mercado e planejar um inovador passo rumo aos dispositivos eletrônicos inteligentes destinados ao consumidor final. Acreditava-se na provável convergência dos computadores com os equipamentos e eletrodomésticos utilizados no dia-a-dia.

Para provar a viabilidade da idéia, 13 pessoas trabalharam durante 18 meses a fim de criar um protótipo que se chamava *StarSeven*. O protótipo era um controle remoto com uma interface gráfica *touchscreen*. O *StarSeven* tinha a habilidade de controlar diversos dispositivos e aplicações. Para o protótipo foi criada, por James Gosling, uma nova linguagem de programação batizada de *Oak* (carvalho).

A fim de encontrar mercado para o *StarSeven* foi construído um projeto demonstrativo chamado *MovieWood*. O *MovieWood* que pode ser comparado aos programas iterativos e também a televisão digital. Mas segundo o site Wikipédia, o mercado não estava preparado para que o usuário pudesse interagir com a programação e com a emissora em uma grande rede de cabos.

Mas a feliz coincidência do estouro da Internet, que promovia uma grande rede de iteratividade, fez com que a *Sun Microsystems* redirecionasse o projeto. Logo ficou a cargo de

James Gosling de adaptar o *Oak* para a Internet e em Janeiro de 1995 foi lançada à nova versão agora rebatizada com o nome de Java. A tecnologia Java havia sido projetada para se dissipar por meio de dispositivos heterogêneos, como a Internet. Agora as aplicações passam a ser executadas dentro de *browsers* nos *Applets* Java e as informações seriam disponibilizadas pela Internet em tempo real. Com o assustador crescimento do número de usuários rapidamente grandes empresas como a IBM anunciaram suporte à tecnologia Java.

Em 2003 a Java atingiu a marca de quatro milhões de usuários em todo o mundo. E a tecnologia tende a continuar crescendo e é hoje um padrão de mercado oferecendo qualidade, performance e segurança ainda sem competidores a altura.

A tecnologia tornou-se popular em sua utilização na Internet e atualmente possui seu ambiente de execução difundido em *web browsers, mainframes, Sos, celulares, palmtops,* dentre outros. [WIK 06] e [DEI 01] e [PAM 06]

2.2.3 Máquina Virtual Java

Diferentemente de sua linguagem originária a tecnologia Java não é compilada e sim interpretada em uma representação intermediária conhecida como *bytecodes*. Os *bytecodes* produzidos pelos compiladores Java podem ser utilizados num processo de engenharia reversa para o programa fonte original.

Alguns profissionais acreditam que por ter essa característica a Java possua um baixo desempenho. Mas após a criação, por parte da *Sun* Microsystems, do compilador *Just-In-Time (JIT)* que analisa e retira os códigos desnecessários aumentando consideravelmente a velocidade da execução essa afirmação cai por terra. Pois os avanços que tem tornado o compilador (*Máquina Virtual Java ou JVM*) mais dinâmico.

Java, atualmente, possui desempenho próximo ao da linguagem C++ graças a otimizações no código como a compilação especulativa e o *HotSpot* da *Sun MicroSystems.* A compilação especulativa aproveita o tempo ocioso do processador para efetuar a pré-compilação dos *bytecodes* para códigos nativos, enquanto o *HotSpot* da *Sun MicroSystems*, armazena informações disponíveis somente em tempo de execução, como por exemplo, número de usuários e memória disponível, para aperfeiçoar o funcionamento da *JVM*.

Logo, devido ao bom e crescente desempenho da Java é comum se encontrar sistemas corporativos e de missão crítica utilizando a tecnologia Java. Segundo informações do site Wikipédia no Brasil a maioria dos bancos utiliza a tecnologia Java para construir seus *home banks*, que possuem milhares de acessos diários e simultâneos. Grandes sites como o eBay utilizam Java para garantir a alta performance exigida pelo seu ramo de atuação.

A *JVM* é a responsável por criar um ambiente multiplataforma, ou seja, para cada plataforma que suporte a tecnologia Java é necessária a criação de uma *JVM* específica que traduza os *bytecodes* para o código nativo. Dentre outras funções, a *JVM* é responsável por carregar de forma segura todas as classes, verificar se os *bytecodes* aderem às especificações e se a integridade e segurança do sistema poderão vir a ser violados.

A figura abaixo ilustra o processo de compilação e execução de um programa Java. A partir de um código Java localizado em um arquivo com a extensão *.java*, o compilador *javac* gera o *bytecode* em um arquivo com a extensão *.class*. Após isso a máquina virtual java executa o *bytecode* e roda o programa que pode ser em ambiente com sistema operacional Windows ou para Celular. [WIK 06] e [DEI 01] e [PAM 06]

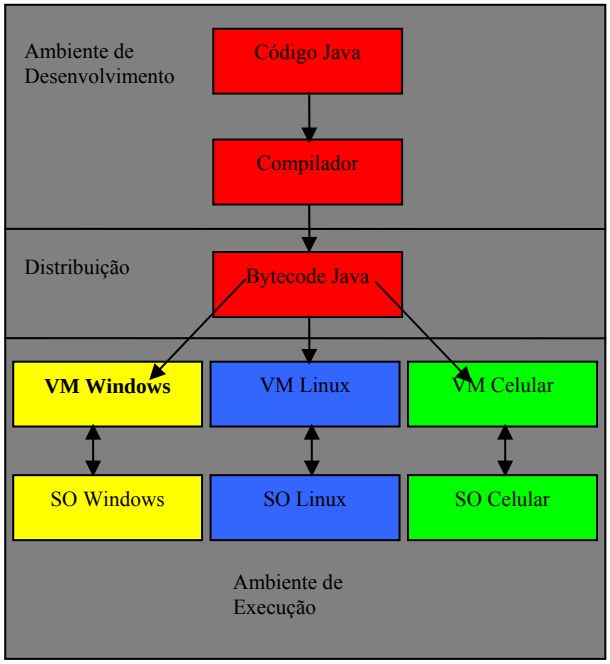

FIGURA 2.1 – Compilação e Interpretação de um programa Java

2.2.4 Java e Suas Edições

A tecnologia Java se divide em três edições [DEI 01] e [PAM 06]:

- **Java 2 Standard (J2SE):** É a tecnologia Java para computadores pessoais e arquiteturas de processamento e memória consideráveis. Munida de várias APIs o J2SE pode ser adquirido por meio de download no site da *Sun microsystems.* O J2SE se divide em:
    - **Java Development Kit (JDK) ou Standard Development Kit (SDK):** pacote para desenvolvimento Java;
    - **Java Runtime Edition (JRE):** uma versão mais leve do JDK por ser preparada para o ambiente de execução, ou seja, executará os sistemas construídos com o SDK.

- **Java 2 Mobile Edition (J2ME):** é a tecnologia Java voltada para dispositivos móveis com limitação de memória ou processamento. Utilizada em sistema como celulares, palmtops, dentre outros. O J2ME se divide em:
    - **Connected Limited Device Configuration (CLDC):** Para dispositivos mais limitados como os celulares;
    - **Connected Device Configuration (CDC):** para dispositivos mais poderosos como os palmtops.
- **Java 2 Enterprise Edition (J2EE):** é a tecnologia para aplicações corporativas. Possui um grande número de APIs, onde o grande foco é a segurança. Ideal para construção de servidores de aplicação, integração de sistemas ou distribuição de serviços para terceiros.

2.2.5 Java e Suas Ferramentas

No diretório de instalação do Java encontra-se uma pasta *bin*. Dentro desta pasta estão localizadas várias ferramentas da linguagem Java. A tabela 2.1 mostra as principais ferramentas estudadas. [PAM 06]

TABELA 2.1 – Ferramentas da Java.

Executável	Nome	Descrição
appletviewer	Java Applet Viewer	Visualizar applets sem navegador.
java	Interpretador Java	Executa as aplicações java compiladas (*bytecodes*).
javac	Compilador Java	Compila os programas Java. Sua saída é em *bytecodes* (*.class*).

javadoc	Documentador	Gera a documentação através dos comentários adicionados ao código.
jar	Compactador	Compacta em um único arquivo.
javap	Decompilador	Exibe o conteúdo do *.class*
jdb	Java Debbugger	Efetua o debug dos programas Java.

2.2.6 Orientação a Objetos

A programação orientada a objetos que é o paradigma dominante atualmente, tem substituído a tradicional técnica programação estruturada criada na década de 70. Java é totalmente orientado a objetos, e é impossível se programar no estilo procedural. Um sistema orientado a objetos é composto por um conjunto de classes e objetos bem definidos que interagem de modo a gerar resultados satisfatórios.

Os programas são constituídos de objetos, com certas propriedades e operações que os objetos podem prover. Na OOP (*Object-Oriented Programming*) a única preocupação é com o que o objeto faz. Não interessa como um objeto é implementado contanto que se saiba o que faz.

A chave para obter-se o máximo de produtividade na OOP é fazer com que cada objeto seja responsável por realizar um jogo de tarefas bem determinadas. Se um objeto realizar tarefas que ele não seja responsável, o mesmo deverá acessar outro objeto que realize as tarefas desejadas.

Em particular, um objeto nunca poderá manipular diretamente uma informação constante em outro objeto. Toda a comunicação deve ser via chamada de método. Por *encapsulamento* (esconder a implementação de um dado para o objeto que a usa) da informação, utilizando a usabilidade, reduz a dependência e o tempo de depuração. Pois o

desenvolvimento e a depuração são simplificados quando se constrói um objeto que executa poucas tarefas, ao contrário de objetos que manipulam várias informações tornando-o extremamente complexo, com várias funções para manipular as informações. [PAM 06] e [DEI 01]

O Vocabulário da OOP

A fim de facilitar o entendimento de algumas terminologias da OOP serão apresentadas. O termo mais importante é a Classe. A classe é um template (molde) de como os objetos atuarão. Quando se constrói um objeto para uma *classe*, deverá ser dito que foi criada uma *instancia.*

Como se pode ver, todo código escrito em Java está dentro de uma classe. A biblioteca básica do Java prove centenas de classes para diversas proposições como o desenvolvimento de interfaces, datas e calendários, dentre outros. Entretanto será necessária a criação, e/ou adaptação, de classes Java para que se possam obter objetos de domínio específico para a aplicação desejada.

O *encapsulamento* é a chave do conceito no uso de objetos. Formalmente, encapsulamento não é mais que esconder a implementação de um dado para o objeto que a usa. Os dados em um objeto são os campos de instancia, e as funções e procedimentos que são operadas por um método. O encapsulamento é a maneira de se obter uma "caixa preta" que consiste na chave para a re-usabilidade.

Uma classe pode ser constituída de outras classes. Quando isso acontece diz-se que a classe estende (*extends*) outra classe. O Java, na realidade, vem de uma "classe base" chamada de *Object*, porque cria todos os objetos. Todas as outras classes estendem dela.

Quando e estende uma classe existente, a nova classe terá todas as propriedades e métodos da classe estendida. A classe poderá prover novos métodos e atributos que serão aplicados somente à criada. O conceito de estender uma classe para obter a outra classe é chamado de *herança (inheritance).* [DEI 01] e [PAM 06]

Na tabela 2.2 são apresentadas definições sucintas sobre a arquitetura de orientação a objetos em Java.

TABELA 2.2 – Termos e definições da arquitetura de orientação a objetos em Java.

Termo	Definição
Pacote	Conjunto de classes e demais arquivos classificados por interesses comuns ou possuem dependência entre si.
Instância - Objeto	Uma variável do tipo de uma classe.
Construtor	Responsável por efetuar a criação e inicialização de uma instância de classe.
Método	Funções providas pela classe.
Hierarquia de classes	Grupo de classes relacionadas por herança.
Classe base	É a “classe pai”, classe que dá origem a todas as outras classes.
Superclasse	É a “classe mãe”, a classe que é estendida por uma determinada classe.
Subclasse	É a “classe filha”, a classe que estendeu uma determinada classe.

2.3 PLATAFORMA J2ME

O Java 2 Plataform, Micro Edition (J2ME) é a plataforma desenvolvida para dispositivos de propósitos específicos como os aparelhos de telefonia móvel, os PDAs, e uma grande escala de outros dispositivos provenientes da tecnologia JAVA. Assim como o J2SE, o J2ME é uma plataforma munida de APIs [WIK 06] (conjunto de rotinas e padrões estabelecidos por um software para utilização de suas funcionalidades por programas aplicativos -- isto é: programas que não querem envolver-se em detalhes da implementação do software, mas apenas usar seus serviços) Java definidas pelo *Java Community Process (JPC)* [JCP 06]. O JCP é um processo formalizado que permite que as partes interessadas se envolvam em definições de versões futuras e funcionalidades da plataforma Java.

Esta comunidade definiu a arquitetura computacional dos pequenos dispositivos. Logo, foi definido como ambiente de execução JAVA, e um conjunto de classes básicas, conhecidas como *core*, que operam sobre cada dispositivo. A primeira abordagem foi denominada de configurations são responsáveis por definir a JVM para um pequeno dispositivo computacional.

A segunda abordagem foi definida como *profile* que consiste em um conjunto de classes que possibilita os desenvolvedores de software implementarem as aplicações de acordo com as características das aplicações dos pequenos dispositivos. Tais características foram batizadas como *MIDP (Mobile Information Device Profile)*. O MIDP oferece recursos como rede, componentes de interface, armazenamento local, dentre outros.

Ainda é importante ressaltar que seção está focada em ressaltar apenas os pontos relevantes a criação da interface a ser manipulada por meio do aparelho celular. [MUC 04]

2.3.1 Configuration

A configuration basicamente disponibiliza bibliotecas básicas e a Virtual Machine (VM), sendo esta a parte mais primitiva e indispensável necessitando estar presente em todo e qualquer dispositivo que suporte tal tecnologia.

A configuration do J2ME mais importante é batizada como CLDC (Connected Limited Device Configuration). A configuration anteriormente citada também é base para outra configuration batizada como CDC (Connected Limited Configuration) a qual prove funcionalidades a mais. Mas como conseqüência requer mais hardware. Tal configuration não será tratada em mais detalhes por não fazer parte do enfoque do projeto.

O CLDC possui as configurações para aparelhos extremamente pequenos, dispositivos de no mínimo 160 KB e algumas dezenas de MHz de processamento. Logo é importante criar códigos enxutos e métodos bem analisados, a fim de ocupar o mínimo de memória e processamento. [MUC 04]

2.3.2 Profiles

Um *profile* consiste em uma série de APIs padrões que combinados com uma configuration disponibiliza um serviço para aplicações possam ser executadas. O *profile* que adota será o MIDP 1.0

Como o mercado em sua maioria trabalha, atualmente, com o MIDP 1.0 e o CLDC 1.0 e ainda essa combinação atende perfeitamente os requisitos será essa a configuração adotada. [MUC 04]

Configuration X Profiles

A fim de esclarecer o que seriam *configurations* e *profiles*, e quais as suas atribuições é exposto aqui uma analogia sobre a construção de um veículo automotivo. Sendo assim, partindo do princípio que os profiles são mais específicos que os configurations, entendem-se:

Em uma fábrica de veículos automotivos é sabido como os carros são construídos e no que consiste um carro. Tal analogia seria chamada de configurations. Caso a analogia seja feita a uma fábrica específica, e os questionamentos relacionados a como os carros são construídos fossem aplicados, se obteria os profiles.

MIDP 1.0 X MIDP 2.0

A principal diferença entre as versões dos profiles está relacionada à segurança oferecida. A versão MIDP 1.0 oferece conexão no CLDC 1.0 por meio do protocolo HTTP, que no caso é desprovido de um dispositivo de segurança. Já a versão MIDP 2.0 oferece uma conexão segura, provida de criptografia, por meio do protocolo HTTPS. Ainda a versão mais atual do profile oferece facilidades como classes de jogos e tratamento de sons. [MUC 04]

2.3.3 Midlet

Ciclo de Vida

Cada dispositivo possui um *Aplication Manager (AM)* responsável pelo controle dos aplicativos a serem instalados, onde e como serão armazenados e como serão executados. A localização das classes de cada aplicativo em um arquivo JAR, sendo que este é acompanhado de um descritor batizado de *JAD*, que possuirá todas as informações.

a) Arquivos .JAD e .JAR

Após o processo de criação do aplicativo, é necessário transformá-lo em um formato compatível com os aparelhos celulares: o formato é o JAR (Java Arquive). O mesmo agrupa todas as classes utilizadas para o desenvolvimento e pode conter vários KB.

Por existir uma grande variedade de aparelhos celulares com características específicas também é gerado um arquivo JAD (Java Application Descriptor), muito menor que o JAR, que possui todas as características da aplicação. Assim, quando se deseja efetuar o download de uma aplicação para o celular, é feito o download do arquivo JAD, e se for verificado a compatibilidade com o modelo a ser instalado, iniciá-se o download do arquivo JAR.

Após a chamada de uma MIDlet, o AM invoca o método startApp(). Em momento de execução caso surja a necessidade de pausar a o MIDlet o método pauseApp() deverá ser invocado. Tal recurso poderá ser solicitado caso no momento de executação aconteça uma chamada. No caso de surgir a necessidade de fechar a aplicação o método destroyApp() deverá ser invocado. [MUC 04]

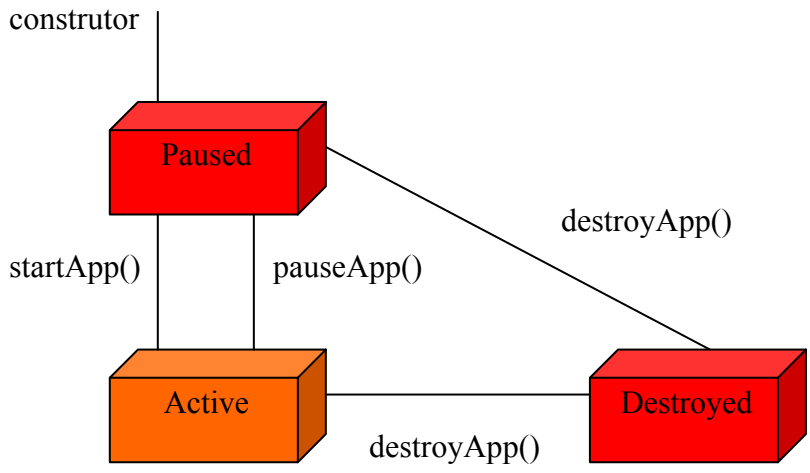

FIGURA 2.2 – Ciclo de vida de um MIDlet.

Interface

Desde que o dispositivo contenha uma VM apropriada, se torna possível executar as MIDlet no mesmo. Porém com a existente diferença nas características de cada dispositivo com tamanho de tela, cores, dentre outros vários aspectos, torna-se bastante difícil desenvolver uma interface apropriada para a utilização por parte do usuário.

O desenvolvimento se torna viável através de uma certa abstração de tela, pois os comandos e inserção de dados são executados por meio dos botões do celular. Logo as aplicações tomam conhecimento desses detalhes em tempo de execução e se adaptam e comportam de maneira apropriada em cada aparelho.

A tela do dispositivo pode ser obtida através de uma instância da classe Display, obtida por meio do método getDisplay(), geralmente contida no método startApp() responsável por inicializar um MIDlet. Nessa instância de Display serão inseridas heranças de Displayable.

Ex.:

```
Public void startApp(){
    Display display = Display.getDisplay(this);
}
```

Logo o ciclo básico pode ser definido por:

- Mostrar um Displayable;
- Ação do usuário;
- Decidir próximo Displayable;
- Repetição do fluxo;

As ações do usuário são gerenciadas por comandos intituladas Commands.

Commands

Possuidores da função de interação com o usuário por parte da aplicação. Estes podem vários tipos: itens de menu, botões, dentre outros. Os commands possuem três propriedades:

- Label: texto mostrado ao usuário;
- Type: tipo do comando;
- Priority: prioridade dos comandos que são utilizados para mostrar os comandos mais utilizados com maior facilidade.

Ex.:

comando = new Command(“OK”, Command.Ok, 0);

Para tornar possível a implementação dos commandos é necessário que a classe herde da classe CommandListener. A classe CommandListener força a implementação do método commandAction que é onde a implementação dos comandos deverá ser executada. Para cada objeto que herda de Displayable existem os métodos addCommand() e removeCommand(), sendo que estes são métodos auto-explicativos.

É extremamente importante utilizar os tipos de commands corretos. Pois os mesmos em tempo de execução possuem a definição do correto posicionamento na tela.

Existem vários tipos de commands, mas neste documento apenas serão citados os utilizados no desenvolvimento da interface de interação com o usuário.

TextBox

Utilizado para entrada de dados texto em celulares. É importante se atentar ao fato de que existe uma dificuldade na entrada de dados texto por parte do usuário por meio do celular. Logo, a utilização de tal recurso deve ser evitada a fim de facilitar a usabilidade da interface.

O construtor de um TextBox é:

public TextBox(String title, String value, int maxLenght, int constraints);

onde:

- “title”: é o label do campo;
- “value”: valor do textbox;
- “maxLenght”: máximo de caracteres permitidos;
- “constraints”: determina se o tipo é NUMERIC, ANY(numérico ou texto) ou PASSWORD.

List

São listas que permitem o usuário selecionar um ou vários elementos de acordo com o tipo escolhido. Existem três tipos de lists são estas: IMPLICIT, EXCLUSIVE e MULTIPLE. O tipo IMPLICIT permite escolher uma opção e clicar no botão padrão de seleção ou em um command. O tipo EXCLUSIVE: muito parecido com uma lista de radio (referencia ao HTML), permite a seleção de uma opção. E finalmente o tipo MULTIPLE: que possui o funcionamento parecido com um CheckList (referencia ao HTML), que permite a seleção de vários itens numa lista.

Quando a lista tornar-se grande demais para a tela, automaticamente a VM criará uma rolagem para possibilitar a visão de toda a lista. Cada item da lista possui um índice. Esse índice inicia em 0.

Alerts

Simplesmente uma exibe uma mensagem informativa ao usuário. Os alerts possuem dois tipos, sendo estes Timed Alerts ou Modal Alert. No primeiro é possível setar um tempo, caso necessário. O segundo aparece e fica aguardando a intervenção do usuário e subdividi-se em vários tipos: Alarm, Confirmation, Error, Info e Warning. O seu construtor de um Alert é:

public Alert(String title, String value, Image image, AlertType type);

Conexões

Sistemas movies geralmente oferecem acesso a dados via conexões OTA(Over The Air), porém essas conexões ainda são lentas, pouco confiáveis e intermitentes. Tal caracterização dificulta a transmissão de dados.

A profile MIDP possui mecanismos genéricos para acesso a rede, sendo que estas são definidas pela Configuration CLDC. Logo por meio do pacote javax.microedition.io. Connection, que é baseado na interface Connection, se torna possível passar uma String de conexão e obter como resposta um Connection.

Como discutido anteriormente neste projeto será utilizada a CLDC 1.0 que prove o padrão HTTP. O HTTP trabalha com o processo response/request incluindo Headers no processo. Os parâmetros enviados na url são codificados antes da transmissão o que garante integridade da informação e padronização. Para isto utiliza-se de URLEncoder, que transforma espaços em sinais de +, a...z, A...z, 0...9, ponto(.), hífen(-), asterisco(*) e Underscore(_) são mantidos, e todo e qualquer outro caractere é convertido em %xy, onde xy é o valor Hexadecimal que representa 8 bits menos significativos do caractere.

b) Protocolo HTTP

HTTP (Hyper Text Transfer Protocol) é o protocolo de transferência de dados utilizado na web e é a base de toda a sua funcionalidade. Arquitetado a partir dos protocolos TCP/IP, sendo também orientado à conexão. O protocolo HTTP permite que os dispositivos troquem informações com os servidores web, efetuando requisições e recebendo respostas por meio de uma conexão internet.

O protocolo HTTP está disponível desde a versão 1.0 do MIDP. Na versão 2.0 foi disponibilizado o recurso de conexão segura HTTPS.

A conexão GET funciona enviando uma url para o método open() da classe Connector da qual irá receber o resultado como um HTTPConnection ou um InputConnection. Sendo assim se torna necessário obter os streams de conexão InputStream e OutputStream e então ler os dados que foram retornados para o servidor e trata-los conforme a necessidade.

Sempre que for estabelecida uma conexão HTTP, é importante se verificar o retorno dos Headers para saber se o retorno está correto, por meio de HTTPConnection.HTTP_OK.

Sempre é necessário ao se criar uma conexão obter uma thread separada. Pois a conexão pode ser tornar lenta ou ainda nem ser concluída. [MUC 04]

2.4 XML

A XML (*eXtensible Markup Language*) é uma linguagem que permite que uma marcação específica seja criada para especificar idéias e posteriormente compartilhá-las. A XML se tornou rapidamente no padrão de intercambio de informações na Web. É constituída de tags (caracterizados pelos identificadores < ... >). Em conjunto as tags são batizadas como "*markup*" ou marcadores. O XML é responsável por identificar os dados, e não como indicá-los. Ou seja, o XML atua como um campo do programa. [DAC 01] e [HAL 00]

2.4.1 TAGS E ATRIBUTOS

As tags podem ser definidas de acordo com a necessidade de seu programa, como exemplificado abaixo:

<mensagem>

<para>exemplo@provedor.com.br</para>

<de>criador@provedor.com.br</de>

<assunto>exemplo</assunto>

<texto>Texto de exemplo</texto>

</mensagem>

Ou então podem também conter atributos, informações adicionais incluídas como parte integrante das *tags*, limitadas pelos símbolos de "<" e ">". Logo, aproveitando o exemplo acima tem-se:

<mensagem

para=exemplo@provedor.com.br

de=criador@provedor.com.br

assunto="exemplo">

<texto>Texto de exemplo</texto>

</ mensagem>

Os atributos contidos nas *tags* são separados por espaços. Os atributos são definidos pelo nome, seguidos do símbolo igual e seu conteúdo seguido limitado por aspas duplas.

O XML exige uma grande atenção no que diz respeito ao seu padrão de formatação exigido. Existem regras severas que determinam como um documento deve estar formatado,

mas uma das mais importantes é o fechamento das *tags.* Em XML não é opcional que uma *tag* aberta anteriormente seja fechada. Segue o exemplo abaixo:

<para>exemplo@provedor.com.br</para>

2.4.2 COMENTÁRIOS EM XML

A fim de acrescentar comentários ao código para que desta forma o mesmo se torne de mais fácil entendimento usa-se a seguinte terminologia:

<!-- comentário -->

O conteúdo inserido nesta *tag* apenas poderá ser visualizado dentro do código do XML.

2.4.3 O PROLOGO DO XML

O prólogo do XML é a parte do documento que precede os dados do mesmo. O prólogo é onde serão incluídas as informações do arquivo. Um arquivo XML **sempre** será iniciado por um prólogo. A mínima declaração exigida é o atributo que indica a versão, mas outros atributos poderão ser acrescentados como pode ser verificado abaixo:

<?xml version=”1.0” encoding=”ISO-8859-1?>

O atributo **version** diz respeito a versão utilizada no arquivo XML. O atributo **encoding** é o identificador de qual decodificador deverá ser usado para decodificar os dados.

2.4.4 VANTAGENS

- A XML é extensível. A possibilidade de criar etiquetas de um modo arbitrário (respeitando sempre as regras), permite adaptar a estrutura de um documento XML a praticamente qualquer situação específica.
- Os documentos XML são auto-descritivos. São, portanto, relativamente fáceis de interpretar, manipular e interrogar. Esta característica pode também revolucionar o modo como as pesquisas são efetuadas na web, permitindo o aparecimento de motores de pesquisa que realizem as pesquisas tendo em conta o significado (contexto) dos dados, em vez de se basearem unicamente na associação de palavras-chave.
- Apesar da sua simplicidade, a XML permite criar estruturas bastante complexas (árvores ou grafos de profundidade arbitrária e, eventualmente, cíclicos e recursivos).
- A XML é extremamente flexível, possibilitando a representação, quer de dados estruturados, quer de dados semi-estruturados. Recorrendo à definição de um esquema, é possível efetuar a validação de documentos. Esta característica revela-se de extrema importância para aplicações em que a verificação estrutural de dados seja vital, como é o caso das bases de dados convencionais.
- O conteúdo de um documento XML pode ser facilmente manipulado pelas aplicações de *software* (recorrendo às APIs existentes), o que torna possível atingir níveis de automação bastante elevados.
- Uma vez que a XML tem uma natureza metalingüística, as organizações podem utilizá-la para desenvolver padrões específicos (novas linguagens baseadas na XML, como, por exemplo, a MathML1), definindo esquemas comuns, de modo a

trocarem, eficientemente, dados entre si. Estes esquemas podem ser disponibilizados na web. O objetivo é utilizar a XML como a "língua franca" para a troca de dados entre os sistemas de informação organizacionais.

- A XML é um padrão aberto. Os documentos XML são independentes das aplicações, dos sistemas operativos, etc. Esta característica pode vir a revolucionar a integração de sistemas heterogêneos.
- Uma vez que o conteúdo de um documento XML está separado da sua apresentação, é possível obter múltiplas perspectivas sobre um mesmo documento XML (recorrendo à XSL).

2.5 APACHE TOMCAT

A explosão do fenômeno chamado Internet, acarretou em pelo menos dois novos desafios: o fato das aplicações passarem a ser distribuídas; e as aplicações passaram a ser formadas de um conjunto de tecnologias e linguagens. [WIK 06] e [APA 06]

Sendo assim, a fim de se adaptar as novas características, as aplicações passaram a ter um contexto previamente definido: o sistema deverá ser distribuído utilizando um modelo cliente-servidor, onde o *HTTP* será o protocolo de comunicação entre as aplicações, o *TCP/IP* será o protocolo de transporte de dados pela rede, dentre outros fatores.

2.5.1 O TOMCAT

O *Tomcat*, desenvolvido pela fundação *Apache*, possibilita a execução de aplicações para a web. Desenvolvido com o enfoque em atender as necessidades da tecnologia JAVA, mais especificamente nas tecnologias *Servlets* e *JSP*. Conhecido por ser um software livre, pode ser utilizado com fins comerciais ou não.

O *Tomcat* está escrito em JAVA, logo necessita que uma versão da tecnologia esteja corretamente instalada, no mesmo computador onde será executado. No entanto, não é suficiente a versão *runtime* estar instalada na máquina, se faz necessário uma versão completa, pois o *Tomcat* necessita compilar os programas escritos em JAVA. Conhecido pela sua robustez e eficiência, o *Tomcat* é muito utilizado em ambientes de produção. [APA 06]

2.6 IDE ECLIPSE

O Eclipse é uma IDE (programa de computador que reúne características e ferramentas de apoio ao desenvolvimento de software com o objetivo de agilizar este processo) *open source*. O projeto teve seu início em novembro de 2001, quando a IBM doou um projeto de 40 milhões de dólares à comunidade de software livre. [STO 02]

O Eclipse é fortemente baseado ao desenvolvimento orientado a *plugins (programas de computador utilizado para adicionar funções a outros programas para provir uma função particular ou muito específica)*. [WIK 06]

Seus três maiores projetos são:

- **The Eclipse Project:** responsável pelo desenvolvimento da IDE Eclipse, a ferramenta de desenvolvimento JAVA (JDT), e o desenvolvimento de *plugins* (JDE);

- **The Eclipse Tools Project:** focado na criação de ferramentas para a plataforma Eclipse. Como uma IDE COBOL, C/C++ e outras;
- **The Eclipse Technology Project:** focado na busca de tecnologias, incubadoras e educação no uso do Eclipse.

A plataforma Eclipse quando combinada com o JDT, oferece muita das características disponibilizadas pelas IDE's comerciais como: complemento de código, ferramenta de debug, dentre outros.

2.6.1 PLUGIN ECLIPSE ME

O *plugin* Eclipse ME é uma ferramenta que ajuda no desenvolvido de programas em J2ME. A mesma é utilizada para simular um aparelho móvel. [ECL 06]

CAPÍTULO 3

ARQUITETURA DO PROJETO DE AUTOMAÇÃO RESIDENCIAL

Neste capítulo são apresentadas a estrutura do projeto e sua organização. Serão detalhados os componentes da aplicação, assim como informações inerentes aonde cada um dos componentes são utilizados e qual a sua função.

3.1 COMPONENTES

O projeto é implementado segundo a abordagem cliente servidor. No lado cliente, faz uso dos seguintes componentes:

- **MS (Mobile Station) –** *Plugin* para o ambiente de desenvolvimento Eclipse, utilizando para desenvolvimento J2ME.
- **Software Cliente (MIDlet)** – responsável por apresentar ao usuário uma interface gráfica baseado na tecnologia J2ME.

No lado do servidor, o projeto faz uso dos seguintes componentes:

- **Servidor Web** – Apache Tomcat 4.1, armazena a aplicação que irá executar no lado do servidor.
- **Software Servidor** - responsável por interpretar requisições enviadas pelo usuário e efetuar a comunicação com o hardware desenvolvido utilizando a tecnologia JAVA. Esse software executa no servidor web.

- **Hardware** – responsável por simular as funções de “ligar” ou “desligar” um sistema.

Na Figura 3.1 é apresentado um diagrama que ilustra os componentes relacionados no item 3.1.

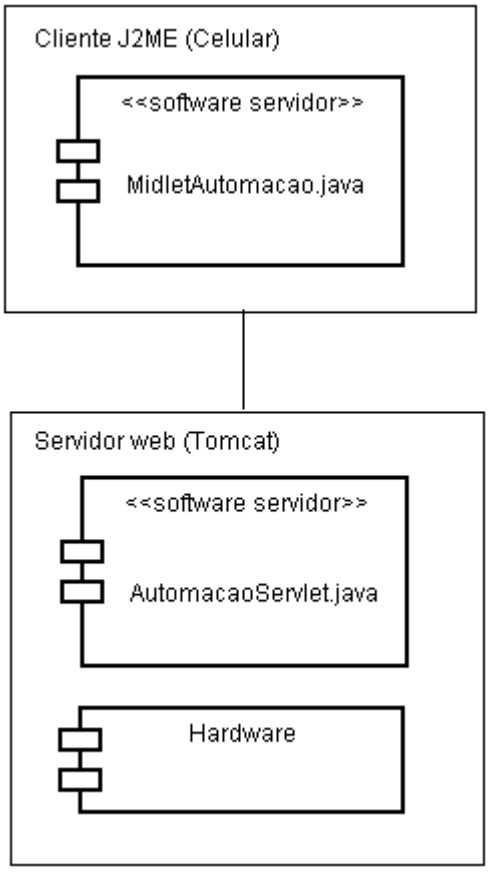

FIGURA 3.1 – Diagrama de Componentes.

3.2 ARQUITETURA DA APLICAÇÃO

Os celulares possuem recursos de memória e de processamento muito limitados, o que dificulta o desenvolvimento de programas mais elaborados. O *Móbile Station* (MS) será utilizado para simular um celular, que envia as solicitações do usuário por meio de uma conexão remota HTTP a fim de utilizar os serviços disponibilizados em um servidor.

A aplicação servidora pode estar disponibilizada em qualquer computador, desde que este tenha um servidor web disponível, o software servidor e possua e um IP conhecido. O software servidor terá como função receber uma solicitação enviada pelo cliente e interpretá-la, para então enviar ao hardware um dado passível de identificação por meio da porta paralela. A porta paralela, dessa maneira é responsável por efetuar a ponte de comunicação entre a aplicação servidora e o hardware.

O hardware é composto por um circuito paralelo que simula a iteração com dispositivos existentes em uma residência, a citar lâmpadas como exemplo. O hardware possui *leds* acoplados a cada ramificação do circuito que possibilitam visualizar o dispositivo referenciado pela solicitação enviada pelo usuário. A arquitetura, brevemente descrita, pode ser visualizada através da Figura 3.2.

FIGURA 3.2 – Arquitetura do projeto.

3.3 APLICAÇÃO CLIENTE

A aplicação cliente é composta de uma interface gráfica (MIDlet) desenvolvida utilizando a tecnologia J2ME, com o auxílio da IDE Eclipse e um plugin denominado EclipseME que simula um celular.

O MIDlet efetuará uma validação de senha para acesso à aplicação. Após a senha ser validada o usuário obterá acesso a alguns recursos da residência, disponibilizados por meio de

um formulário previamente definidos em sistema. Este recurso possui a função de interagir com o usuário a fim de que este possa efetuar uma determinada solicitação dentro das disponibilizadas pelo sistema.

Após a solicitação do usuário, o MIDlet efetuará uma conexão com a aplicação executando no servidor por meio do protocolo HTTP. Durante o desenvolvimento do projeto, o servidor executava na mesma máquina que a aplicação cliente, e dessa forma seu endereço IP é *127.0.0.1* e a porta para acesso ao servidor web é a 8080. A aplicação cliente deverá ainda conhecer o contexto onde foi instalado/codificado o Servlet (parte integrante da aplicação servidora), que consiste no nome do Servlet seguido do comando a ser executado, identificado por meio de um parâmetro de nome "msg", como ilustrado abaixo:

```
HttpConnection com = (HttpConnection) Connector.open(
        http://127.0.0.1:8080/automacao/AutomacaoServlet?msg=”ligar”);
```

Após efetuar a tentativa de conexão, será validado o status da autenticação para que seja verificado se algum erro ocorreu. O status da conexão será disponibilizado para o usuário por meio de uma mensagem apresentada para o usuário. Na Figura 3.3 é demonstrado o fluxo de uma solicitação de ação efetuada pelo cliente.

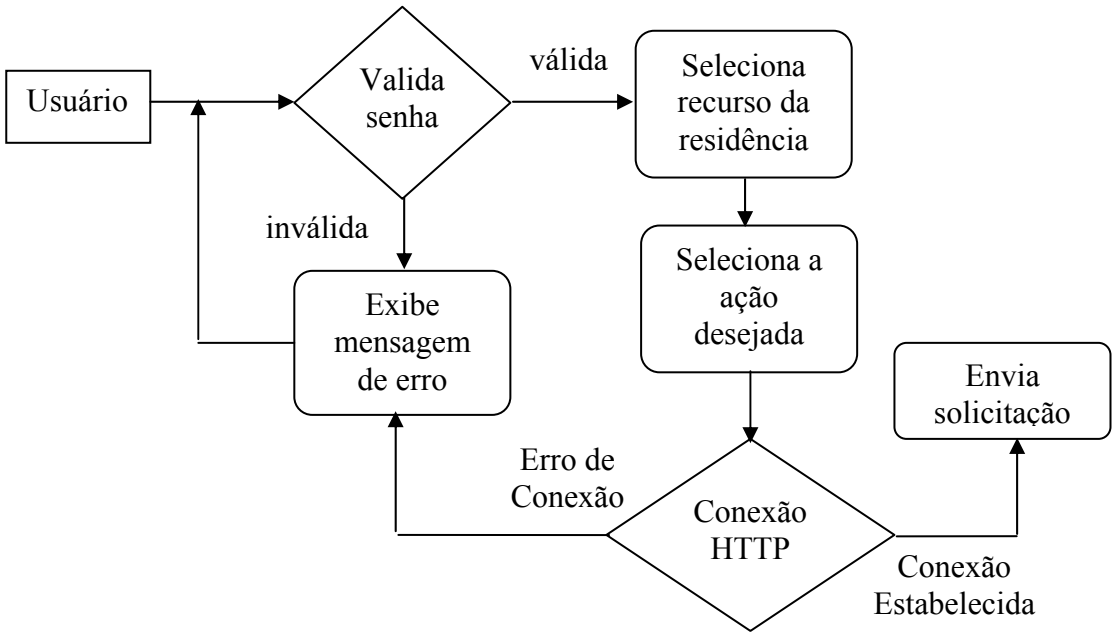

FIGURA 3.3 – Diagrama de fluxo de ação solicitada pelo cliente.

3.4 APLICAÇÃO SERVIDORA

A aplicação servidora é constituída de um Servlet, um programa Java que estende a funcionalidade de um servidor web gerando conteúdo dinâmico e interagindo com o cliente. O contêiner web utilizado para executar o Servlet é disponibilizado pelo o Apache Tomcat, que deverá estar devidamente instalado no computador usado como servidor. O Servlet possuirá a função de receber todas as requisições vindas do celular.

A aplicação servidora possuí ainda o arquivo *web.xml*. Este terá como conteúdo informações que detalham o funcionamento do Servlet. Por se tratar de um arquivo XML, sua estrutura é dividida em campos (*tags*). No mesmo estarão às informações relacionadas ao

nome do Servlet (utilizado para referenciá-lo) e o nome da classe que possui o Servlet. O XML possuirá também o caminho onde o Servlet foi gravado.

O Servlet está incumbido de interpretar a solicitação do cliente e efetuar o repasse da solicitação para um programa desenvolvido com a linguagem C. A comunicação entre as duas linguagens será implementada por meio da tecnologia Java *JNI* (*JAVA Native Interface*), que permite a uma aplicação Java chamar funções disponibilizadas em um arquivo ".dll". Por meio de uma *dll* desenvolvida na linguagem C inserida ao JDK do JAVA, o Servlet fará toda a comunicação com a porta paralela LPT1. O Servlet enviará um sinal para o pino da porta que estiver interligado com o circuito desejado.

A interpretação efetuada pela Servlet consiste em receber um parâmetro enviado pela aplicação cliente e identificar qual dos pinos da porta paralela faz referencia ao circuito desejado. Ainda enviar um bit (sem sinal) para o pino para que assim a ação desejada possa ser visualizada através do hardware.

Na figura 3.4 é demonstrada a funcionalidade executada pela aplicação servidora, (servlet) que executa no servidor tomcat.

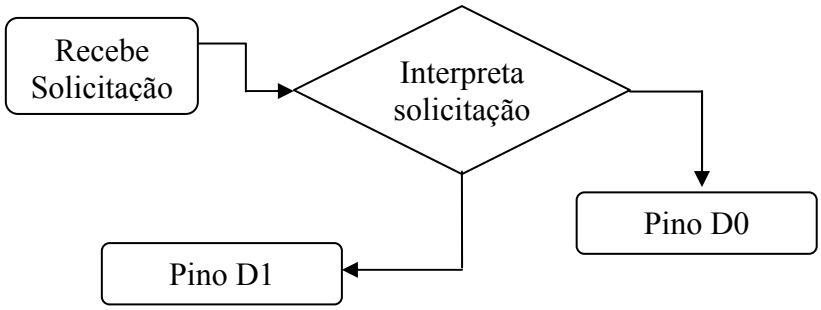

FIGURA 3.4 – Diagrama de fluxo da aplicação servidora.

3.5 HARDWARE

O hardware será constituído de um cabo paralelo modelo DB25 (maiores detalhes, vide no anexo 1) que estará conectado a saída da porta paralela de um computador através do conector macho (Figura 3.5) do cabo. Os circuitos serão compostos de dois resistores de 470 Ω (ohms) e dois interligados. Cada circuito estará ligado a dois pinos da porta paralela, sendo que um pino efetuará o envio de dados para o hardware e o outro funcionará como aterramento. Na Figura 3.6 o circuito é apresentado.

FIGURA 3.5 – Exemplo de um conector macho do cabo DB25.

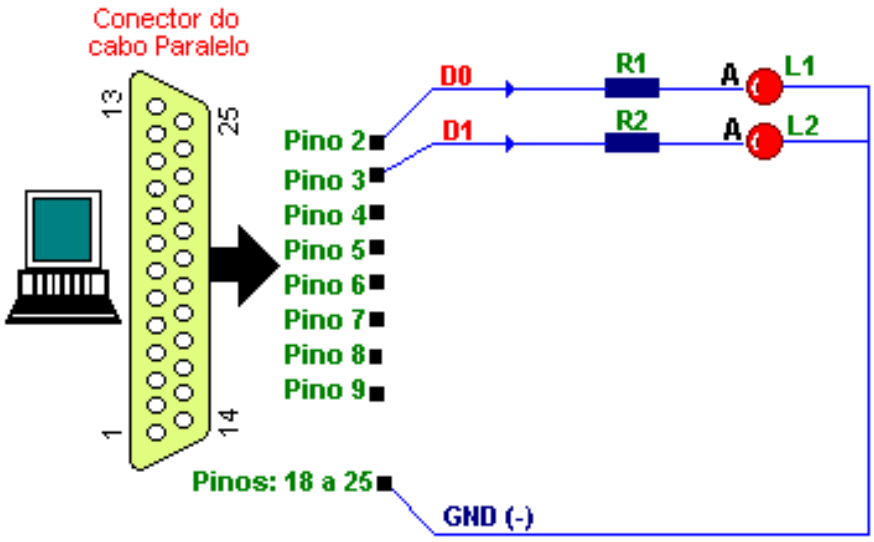

FIGURA 3.6 – Circuito paralelo.

O hardware será responsável por demonstrar através de simulação o acesso aos diferentes dispositivos constantes em uma residência.

CAPÍTULO 4

FUNCIONAMENTO DO PROJETO DE AUTOMAÇÃO RESIDENCIAL

A proposta inicial deste projeto, era produzir um protótipo que com poucas adaptações pudesse ser disponibilizado para comercialização, por meio de uma parceria com uma operadora de telefonia móvel. Mas o desconhecimento em relação aos custos de testes, no que diz respeito a aparelhos e ao acesso web/wap, foi fator determinante para uma mudança na essência do projeto que propunha a instalação de uma interface ao móvel.

Ainda, a superficial avaliação da utilização do IP como item essencial na comunicação entre o móvel e o servidor, que no caso da interface estar instalada no móvel torna-se indispensável à posse de um IP real, ou seja, prioritário.

Desta forma, o projeto se limitou a apresentar a possibilidade de se criar um serviço demonstrando toda a arquitetura básica para que se possa criar um serviço de automação residencial a ser disponibilizado por uma operadora de telefonia móvel. Abrangendo desde o software de interação com o usuário até a comunicação com um hardware desenvolvido com a proposta de ilustrar o serviço.

4.1 Resultados e Testes

A aplicação é iniciada através da IDE Eclipse utilizando o plugin instalado no mesmo EclipseME. O plugin simula um ambiente idêntico ao que um celular proporcionaria como posse ser visualizado na Figura 4.1. A aplicação poderá ser encerrada a qualquer momento clicando no botão que se encontra no teclado identificado por um fone na cor vermelha.

FIGURA 4.1 – Simulador do ambiente de um celular.

Depois de iniciada a aplicação, é requerida uma validação de senha. Sendo que a mesma está cadastrada em sistema. O usuário deverá digitar uma senha utilizando o teclado e clicar no botão OK que se encontra na parte inferior direita da tela. Na Figura 4.2 é apresentada a tela onde será requerido que o usuário digite a senha a ser validada.

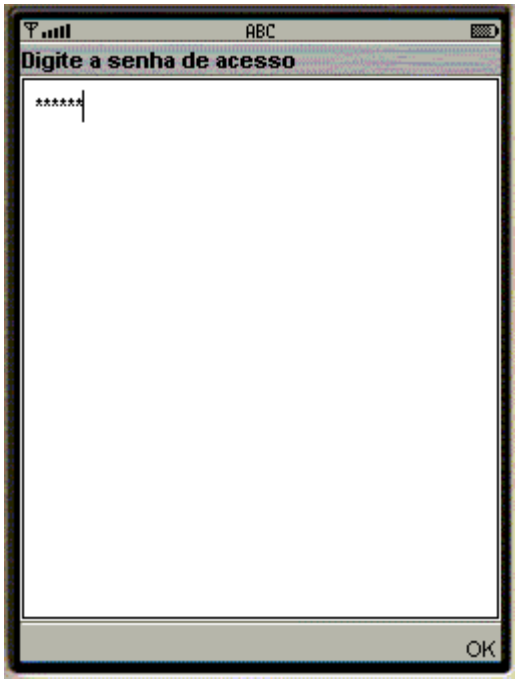

FIGURA 4.2 – Passo para digitação da senha requerida.

No caso de a senha digitada ser inválida, o MIDlet exibirá uma tela de erro onde constará a mensagem “Acesso não autorizado!” e retornará para a tela onde o usuário poderá tentar novamente obter acesso ao sistema. Na Figura 4.3 é apresentada a tela que será exibida em caso de erro ao validar a senha de acesso ao sistema.

FIGURA 4.3 – Mensagem de erro ao tentar acessar o sistema.

Se o usuário digitar a senha válida será exibida uma tela onde constarão todos os dispositivos existentes na residência em questão como mostra a Figura 4.4. Novamente utilizando o teclado, o usuário deverá selecionar o recurso dispositivo desejado e selecionar o botão continuar.

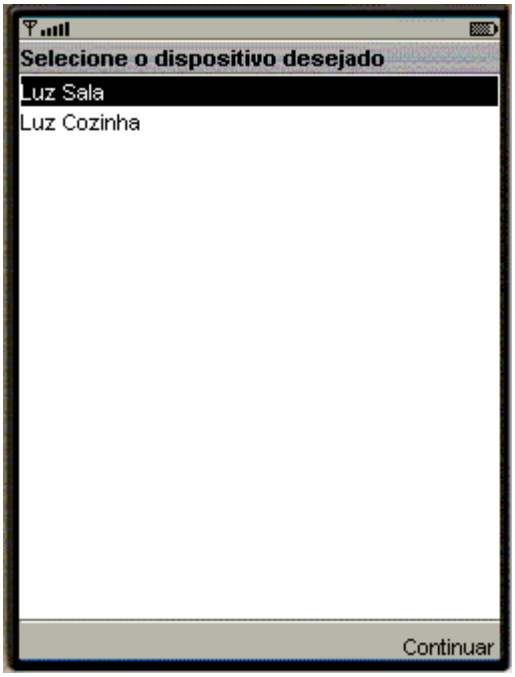

FIGURA 4.4 – Lista de dispositivos da residência.

Logo após, uma nova tela será disponibilizada contendo as ações possíveis, onde o usuário poderá selecionar uma delas por meio do teclado e clicar no botão “Concluir” ou então clicar no botão “Voltar” para efetuar a seleção de um novo dispositivo. Na Figura 4.5 é ilustrado uma tela das ações disponíveis.

FIGURA 4.5 – Lista de ações disponíveis.

Antes da conexão com o Servlet, o MIDlet solicita ao usuário a confirmação por parte do mesmo para estabelecer a conexão, como mostrado na Figura 4.6. É importante frisar que neste passo citado acontece apenas no simulador.

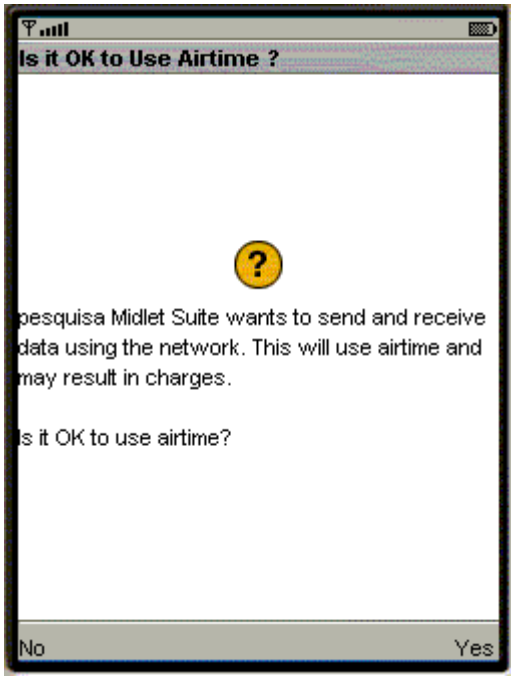

FIGURA 4.6 – Autorização de conexão.

Caso aconteça um erro de conexão, uma mensagem de erro “Erro de conexão!” é mostrada para o usuário como pode ser visualizado através da Figura 4.7. Após apresentar a mensagem, o sistema retorna para tela, onde é possível a seleção de dispositivo da residência.

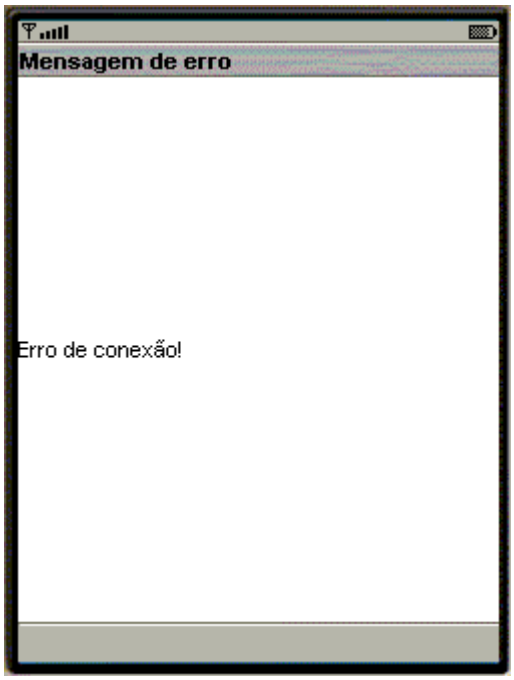

FIGURA 4.7 – Mensagem de erro de conexão.

Se a conexão for estabelecida com sucesso, a mensagem “Solicitação executada com sucesso!” é apresentada e o sistema retorna para a tela que possibilita a seleção de um dispositivo da residência. Na Figura 4.8 é ilustrada a mensagem de sucesso apresentada para o usuário.

FIGURA 4.8 – Mensagem de sucesso ao solicitar ação.

Antes de apresentar a mensagem de sucesso ao usuário, o MIDlet (depois de a conexão ser estabelecida) envia para o Servlet (componente da aplicação servidora) a ação desejada. O Servlet recebe a ação e interpreta de forma a criar um dado capaz de ser interpretado pelo hardware. Paralelamente, a apresentação da mensagem ao usuário, o hardware será acionado e o led correspondente ao dispositivo desejado será ligado ou desligado, de acordo com a solicitação.

CAPÍTULO 5

CONCLUSÕES

Aproveitando o crescimento do mercado de telefonia móvel no Brasil, a proposta aqui apresentada, demonstra a criação de um serviço de automação residencial a ser disponibilizado por uma operadora de telefonia móvel utilizando um celular.

O MIDlet (software para o celular) a ser instalado na estação móvel foi desenvolvido em uma versão de *profile* inicial, ou seja a MIDP 1.0 que apesar ser mais limitada que sua predecessora atende a todos os modelos de aparelhos celulares que possuem suporte a tecnologia JAVA.

A utilização de uma conexão com um servidor foi uma solução muito interessante para se criar um serviço com fins comerciais. Pois através de uma conexão por meio do protocolo HTPP, o usuário pode obter o acesso aos dispositivos de sua residência, de qualquer lugar, desde que o servidor esteja em funcionamento e a operadora dê suporte na área onde o usuário se encontra.

O MIDlet foi desenvolvido atendendo a uma limitação estabelecida pelo móvel que é o recurso de memória e processamento limitado. A fim de atender a essa necessidade o MIDlet foi codificado de maneira enxuta, utilizando os conceitos de orientação objeto e reutilização. Fazendo com que o arquivo gerado seja menor e logo ocupe menos espaço na limitada memória do aparelho celular.

A fim de criar uma aplicação servidora, foram utilizados programas que seguem uma tendência de mercado, o projeto open source. Tal solução foi adotada devido ao fato de diminuir os custos do projeto. Buscando o desenvolvimento de sistemas com máxima eficiência e menor custo, assim como prega a engenharia.

Sendo assim, foi utilizado o software apache tomcat para que se pudesse criar um ambiente servidor. Este software possibilitou sem nenhum custo a criação de uma aplicação servidora que é responsável em efetuar a ponte entre o celular e o hardware de controle aos dispositivos residenciais.

Apesar do projeto se basear na utilização de softwares livres, esta característica não implica em nenhuma desvantagem para o projeto. Pois, os mesmos, são produtos utilizados em larga escala no mercado por grandes corporações, a citar Banco do Brasil. São produtos robustos e com grande suporte disponibilizado pela comunidade integrante do projeto do software livre.

A tecnologia JAVA foi escolhida justamente pela sua portabilidade. Pois apesar de a plataforma utilizada tenha sido o Windows, o JAVA não encontraria problemas em se adequar a uma outra plataforma devido a está característica.

É evidente que para que o projeto atinja um nível de excelência, a ponto de ser disponibilizado como serviço, serão necessárias várias evoluções. Mas, a proposta aqui pretendida foi atendida e demonstrada em sua totalidade. Ou seja, a criação de uma comunicação do móvel com uma determinada residência.

Com isso esse projeto vem agregar grande valor a futuros projetos baseados na disponibilização de serviços ao mercado de telefonia móvel com foco na automação residencial. Este criou toda a arquitetura necessária para projetos a criação do serviço utilizando várias tecnologias de mercado vislumbrando as tecnologias do futuro como a automação residencial.

5.1 TRABALHOS FUTUROS

Existem vários aspectos que sugerem evolução na arquitetura proposta. Deste a evolução da versão do *profile* MIDP 1.0 para a MIDP 2.0 a real aplicação da arquitetura a um hardware de controle que proporcionaria uma real automação residencial.

A evolução do MIDlet sugere a utilização do MIDP 2.0 o que acarretaria na possibilidade da utilização do protocolo HTTPS. Mas, é importante frisar que esta evolução deve ser bem avaliada. Pois esta mudança diminuiria a portabilidade do sistema.

Ainda seguindo a tendência de evolução do MIDlet, seria acoplar a arquitetura um banco de dados que estaria alojado na aplicação servidora a fim de possibilitar o armazenamento de senhas, o armazenamento dos dispositivos da residência possibilitando a criação de um cadastro dos dispositivos. Pois, como frisado várias vezes no decorrer do projeto o celular possui recursos limitados de memória e processamento.

Existe também a necessidade de criação de um pacote de instalação tanto para a aplicação servidora quanto para a aplicação cliente. Tal evolução facilitaria a comercialização do serviço.

E finalmente, a criação de um hardware que proporcione o real controle dos dispositivos constantes na residência.

Desta forma, seguindo esse padrão de evoluções pertinentes ao projeto teria-se um serviço completo a ser disponibilizado e avaliado pelo mercado.

ANEXO 1

CONECTOR DB25

O conector DB25 fica localizado na parte de trás do gabinete do computador, e é através deste, que o cabo paralelo se conecta ao computador para poder enviar e receber dados. No DB25, um pino está em nível lógico 0, quando a tensão elétrica está entre 0 e 0,4V. Entretanto, um pino está em nível lógico 1, quando a tesão elétrica está entre 3.1 e 5V. [MES 04]

A Figura 6.1 demonstra a conectora fêmea do DB25, ponto onde o cabo DB25 será conectado.

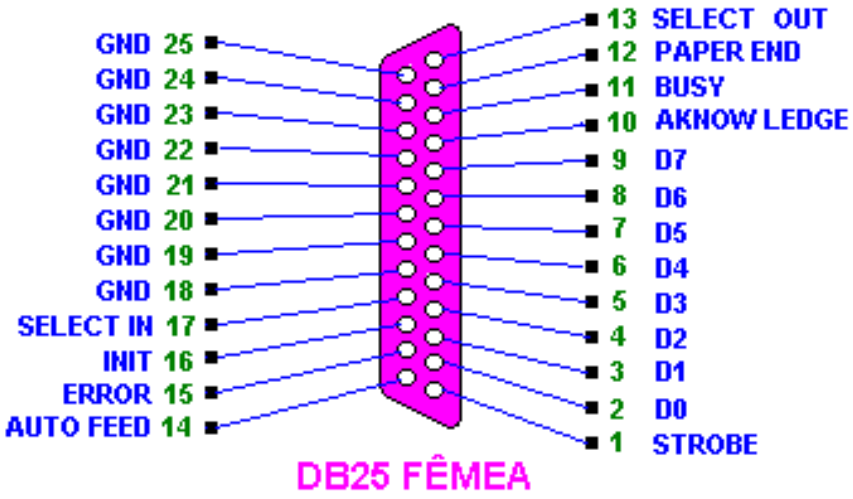

FIGURA 6.1 – Conectora fêmea DB25.

Cada bit do byte está relacionado com um pino do DB25 e um fio do cabo paralelo fisicamente. Ao enviar um byte que o bit esteja ligado ou desligado, o *led* acenderá ou pagará conforme o estado do bit conforme demonstrado na Tabela 6.1.

TABELA 6.1 – Seqüência de bits.

Decimal	Binário	Pino/Fio Ativo 5V
1	0000000**1**	2 - D0
2	000000**1**0	3 - D1
4	00000**1**00	4 - D2
8	0000**1**000	5 - D3
16	000**1**0000	6 - D4
32	00**1**00000	7 - D5
64	0**1**000000	8 - D6
128	**1**0000000	9 - D7

ANEXO 2

CODIFICAÇÃO

Aqui serão apresentados detalhes inerentes a implementação, onde poderão ser verificadas as técnicas de boa programação. Ainda, é apresentada a utilização dos componentes específicos de cada linguagem de programação.

7.1 CODIFICAÇÃO SOFTWARE CLIENTE

```
import javax.microedition.io.ConnectionNotFoundException;
import javax.microedition.io.Connector;
import javax.microedition.io.HttpConnection;
import javax.microedition.lcdui.Alert;
import javax.microedition.lcdui.Choice;
import javax.microedition.lcdui.Command;
import javax.microedition.lcdui.CommandListener;
import javax.microedition.lcdui.Display;
import javax.microedition.lcdui.Displayable;
import javax.microedition.lcdui.List;
import javax.microedition.lcdui.TextBox;
import javax.microedition.lcdui.TextField;
import javax.microedition.midlet.MIDlet;
```

```
public class MidletAutomacao extends MIDlet implements CommandListener, Runnable {

    private Display tela;
    private TextBox text;
    private Command comandoVoltar;
    private Command comandoConcluir;
    private Command comandoValidar;
    private Command comandoContinuar;
    private List menu;
    private List exclusiva;
    private TextBox password;

    public MidletAutomacao(){
        //
    }

    protected void startApp(){
        setTela(Display.getDisplay(this));
        getTela().setCurrent(getPassword());
    }
```

```
	protected void pauseApp() {
		// Auto-generated method stub
	}

	protected void destroyApp(boolean arg0) {
		// Auto-generated method stub
	}

	public void commandAction(Command c, Displayable d) {
		if(c == getComandoValidar()){
			validarSenha();
		}
		if (c == getComandoVoltar()) {
			getTela().setCurrent(getMenu());
		}
		if (c == getComandoConcluir()) {
			enviaDados();
		}
		if (((c == List.SELECT_COMMAND) || (c == getComandoContinuar()))&&
(d == getMenu())){
			String[] exclusivaElementos = {"Ligar", "Desligar"};
			if(getMenu().getSelectedIndex() == 0){
				setExclusiva(new List("Luz Sala", Choice.EXCLUSIVE,
exclusivaElementos, null));
			}else if(getMenu().getSelectedIndex() == 1){
```

```
                                setExclusiva(new List("Luz Cozinha", Choice.EXCLUSIVE,
exclusivaElementos, null));
                        }
                        getExclusiva().addCommand(getComandoVoltar());
                        getExclusiva().addCommand(getComandoConcluir());
                        getExclusiva().setCommandListener(this);
                        getTela().setCurrent(getExclusiva());
                }
        }

        protected void validarSenha() {

                if(isPossuiDado(getPassword().getString()) &&
"123456".equals(getPassword().getString())){
                        getTela().setCurrent(getMenu());
                } else {
                        getPassword().setString("");
                        Alert alerta = getAlerta("Acesso não autorizado!", 5000);
                        getTela().setCurrent(alerta, getPassword());
                }


        }

        private Alert getAlerta(String mensagemErro, int timeOut) {
```

```
        Alert alerta = new Alert("Mensagem de erro", mensagemErro , null, null);
        alerta.setTimeout(timeOut);
        return alerta;
}

private boolean isPossuiDado(String informacao) {
        return informacao != null && !"".equals(informacao);
}

protected void enviaDados() {
        new Thread(this).start();
}

protected Command getComandoConcluir() {
        if (this.comandoConcluir == null) {
                this.comandoConcluir = new Command("Concluir",Command.OK,1);
        }
        return this.comandoConcluir;
}

protected void setComandoConcluir(Command selecao) {
        this.comandoConcluir = selecao;
}

protected Command getComandoVoltar() {
```

```
        if (this.comandoVoltar == null) {
                this.comandoVoltar = new Command("Voltar",Command.BACK,0);
        }
        return this.comandoVoltar;
}

protected void setComandoVoltar(Command voltar) {
        this.comandoVoltar = voltar;
}

protected TextBox getText() {
        if (this.text == null) {
                this.text = criaText();
        }
        return this.text;
}

private TextBox criaText() {
        TextBox textBox = new TextBox("Result","",200,TextField.ANY);
        textBox.addCommand(getComandoVoltar());
        textBox.setCommandListener(this);
        return textBox;
}

protected void setText(TextBox t1) {
```

```
        this.text = t1;
}

protected Display getTela() {
        return tela;
}

protected void setTela(Display tela) {
        this.tela = tela;
}

protected List getExclusiva() {
        return exclusiva;
}

protected void setExclusiva(List exclusiva) {
        this.exclusiva = exclusiva;
}

protected List getMenu() {
        if (this.menu == null) {
                this.menu = criaMenu();
        }
        return this.menu;
}
```

```
        private List criaMenu() {
                String[] menuElementos = {"Luz Sala", "Luz Cozinha"};
                List novoMenu = new List("Selecione o dispositivo
desejado",Choice.IMPLICIT,menuElementos,null);
                novoMenu.addCommand(getComandoContinuar());
                novoMenu.setCommandListener(this);
                return novoMenu;
        }

        protected void setMenu(List menu) {
                this.menu = menu;
        }

        public void run() {
                try{
                        HttpConnection con = (HttpConnection) Connector.open(

        "http://127.0.0.1:8080/automacao/AutomacaoServlet?msg=" +
getExclusiva().getString(getExclusiva().getSelectedIndex()) + "&dispositivo="+
getMenu().getSelectedIndex());
                        int status = -1;
                        status = con.getResponseCode();
                        String situacao = (status != HttpConnection.HTTP_OK?"Erro ao
executar solicitação!":"Solicitação executada com sucesso!");
```

```
                getText().setString(situacao);
                Alert alerta = getAlerta(situacao, 5000);
                getTela().setCurrent(alerta, getMenu());
            } catch(ConnectionNotFoundException e) {
                Alert alerta = getAlerta("Erro de conexão!", 5000);
                getTela().setCurrent(alerta, getMenu());
                e.printStackTrace();
            } catch(Exception e) {
                e.printStackTrace();
            }
        }

        protected TextBox getPassword() {
            if (this.password == null) {
                this.password = criaPassword();
            }
            return this.password;
        }

        private TextBox criaPassword(){
            TextBox password = new TextBox("Digite a senha de acesso", "", 6,
TextField.PASSWORD);
            password.addCommand(getComandoValidar());
            password.setCommandListener(this);
            return password;
```

```
    }

    protected void setPassword(TextBox password) {
        this.password = password;
    }

    protected Command getComandoValidar() {
        if (this.comandoValidar == null) {
            this.comandoValidar = new Command("OK", Command.OK, 1);
        }
        return this.comandoValidar;
    }

    protected void setComandoValidar(Command comandoOk) {
        this.comandoValidar = comandoOk;
    }

    protected Command getComandoContinuar() {
        if (this.comandoContinuar == null) {
            this.comandoContinuar = new Command("Continuar", Command.OK,
1);
        }
        return this.comandoContinuar;
    }
```

```
        protected void setComandoContinuar(Command comandoContinuar) {
                this.comandoContinuar = comandoContinuar;
        }

}
```

7.2 CODIFICAÇÃO SOFTWARE SERVIDOR

```
import java.io.DataOutputStream;
import java.util.HashMap;
import java.util.Map;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import com.XpComm;

public class AutomacaoServlet extends HttpServlet{

        private static final long serialVersionUID = 1L;
        private Map mapaDeDispositivos;

        protected void doGet(HttpServletRequest request, HttpServletResponse response) {
                try {
                        response.setContentType("application/octetc-stream");
                        DataOutputStream dataOut = new
DataOutputStream(response.getOutputStream());
                        String dados = request.getParameter("msg");
```

```
                    int pino = 0;
                    if("Ligar".equals(dados)){
                        String dispositivo = request.getParameter("dispositivo");
                        pino = Integer.parseInt((String)
getMapaDeDispositivos().get(dispositivo));
                    }
                    XpComm.carregaDll(pino);
                    if ("PegarDados".equals(dados)) {
                        dataOut.writeUTF("Resposta Enviada!");
                    } else {
                        System.out.println("Mensagem Recebida: " + dados);
                    }
                } catch (Exception e) {
                    System.out.println("Erro de IO");
                }
            }

            private Map getMapaDeDispositivos(){
                if (this.mapaDeDispositivos == null) {
                    this.mapaDeDispositivos = criaMapaDeDispositivos();
                }
                return this.mapaDeDispositivos;
            }

            private Map criaMapaDeDispositivos() {
```

```
        HashMap hash = new HashMap();
        hash.put("0", "1");
        hash.put("1", "2");
        return hash;
    }
}
```

7.2.1 SOFTWARE COMUNICAÇÃO COM A DLL

```
public class XpComm {

    // Implementado na biblioteca dinâmica.
    public static native int parallelComm(int valor);

    public static void carregaDll(int pino){
        System.loadLibrary("XpCommunication");
        int retorno = parallelComm(pino);
        System.out.println("Retorno = " + getMensagem(retorno));
    }

    private static String getMensagem(int valor){
        if(valor < 0){
            return "Erro ao executar a ação solicitada!";
        }
        return "Ação executada com sucesso!";
    }
}
```

7.2.2 – IMPLEMENTAÇÃO DA DLL

```
#include "XpCommunication.h"
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <windows.h>  //Necessário para: LoadLibrary(), GetProcAddress() e HINSTANCE.

//Declaração dos ponteiros para função.
typedef short _stdcall (*PtrInp)(short EndPorta);
typedef void _stdcall (*PtrOut)(short EndPorta, short valor);

JNIEXPORT int JNICALL Java_com_XpComm_parallelComm(JNIEnv *jniEnv, jclass
jClass, jint valorByte)
{
  HINSTANCE hLib; //Instância para a DLL inpout32.dll.
  PtrInp inportB; //Instância para a função Imp32().
  PtrOut outportB;//Instância para a função Out32().
  unsigned char pino;

  //Carrega a DLL na memória.
  hLib = LoadLibrary("inpout32.dll");

  if(hLib == NULL) //Verifica se houve erro.
  {
```

```
  printf("Erro. O arquivo inpout32.dll não foi encontrado.\n");
  return -1;
}

//Obtém o endereço da função Inp32 contida na DLL.
inportB = (PtrInp) GetProcAddress(hLib, "Inp32");

if(inportB == NULL) //Verifica se houve erro.
{
  printf("Erro. A função Inp32 não foi encontrada.\n");
  return -1;
}

//Obtém o endereço da função Out32 contida na DLL.
outportB = (PtrOut) GetProcAddress(hLib, "Out32");

if(outportB == NULL) //Verifica se houve erro.
{
   printf("Erro. A função Out32 não foi encontrada.\n");
   return -1;
}
//Uso das funções outportB() e inportB():
pino = (unsigned char)valorByte;
printf("Pino ativo = %u\n", inportB(0x378));
outportB(0x378,pino); //Ativa o pino desejado.
```

```
   printf("Pino ativo = %u\n", inportB(0x378));
   FreeLibrary(hLib); //Libera memória alocada pela DLL.
   return(0);
}

#ifndef _DLL_H_
#define _DLL_H_

#if BUILDING_DLL
# define DLLIMPORT __declspec (dllexport)
#else /* Not BUILDING_DLL */
# define DLLIMPORT __declspec (dllimport)
#endif /* Not BUILDING_DLL */

#include <jni.h>
#ifdef __cplusplus
extern "C" {
#endif
/* Class: XpCommunication
* Method: parallelComm
* Signature: ()V
*/
JNIEXPORT int JNICALL Java_com_XpComm_parallelComm(JNIEnv *, jclass, jint);
#ifdef __cplusplus
}
```

```
#endif

#endif /* _DLL_H_ */
```

REFERÊNCIAS BIBLIOGRÁFICAS

Apache; **The Apache Software Foundation**. copyright 1999-2006 ©. Disponível em: <http://tomcat.apache.org>. Acesso em: Maio de 2006.

Morais, M. 2005 é recordista no número de novos assinantes na telefonia móvel, que somam mais de 86,2 milhões. **Anatel – Agência Nacional de Telecomunicações**, Janeiro de 2006. Disponível em: <http://www.anatel.gov.br/biblioteca/Releases/2006/release_16_01_2006.pdf>. Acesso em: Maio de 2006.

Bettoni, R.; Automação Residencial um sonho que começa a se materializar. **Portal da Automação**, Setembro de 2003. Disponível em: <http://www.portaldaautomacao.com.br/artigo_012.asp >. Acesso em: Junho de 2006.

Daconda, M; Saganich, A; **XML Development with JAVA 2**. Sam´s publishing, 2001. p. 146 -179

Deitel, H. M; Deitel, P. J.; **JAVA como programar**. 3ª ed. Bookmam: Porto Alegre, 2001.

EclipseME; **EclipseME Home Page**. Disponível em: <http://eclipseme.org>. Acesso em: Maio de 2006.

Hall, M; **More Servlet and JavaServer Pages**. Prentice Hall PTL, 2000. p. 277 – 375

JCP; **JAVA Community Process,** copyright 1996-2006 ©. Disponível em: <http://www.jcp.org/en/home/index>. Acesso em: Maio de 2006.

Messias, A. R.; O Conector DB25. **RogerCom Pesquisa e Desenvolvimento.** Copyright 1999-2004 ©. Disponível em: <http://www.rogercom.com>. Acesso em: Junho 2006.

Muchow, J. W; **Core J2ME: Tecnologia e MIDP**. 1ª ed. Makron Books, 2004.

Pamplona, F.V; O que é JAVA. **Site JavaFree.org.** Janeiro de 2006. Disponível em: <http://www.javafree.org/content/view.jf?idContent=84>. Acesso em: Maio de 2006.

Pamplona, F.V; Orientação a Objetos. **Site JavaFree.org.** Janeiro de 2006. Disponível em: < http://www.javafree.org/content/view.jf?idContent=86>. Acesso em: Maio de 2006.

PUC Rio; História e Evolução dos Telefones Celulares, 2002. Disponível em: <http://www.maxwell.lambda.ele.puc-rio.br/cgi-bin/PRG_0599.EXE/6705_3.PDF?NrOcoSis=19091&CdLinPrg=pt>. Acesso em: Maio de 2006.

Wikipédia; Apache Tomcat. **Wikipédia a enciclopédia livre**, Maio de 2006. Disponível em: < http://pt.wikipedia.org/wiki/API>. Acesso em: Maio de 2006.

Wikipédia; API. **Wikipédia a enciclopédia livre**, Maio de 2006. Disponível em: < http://pt.wikipedia.org/wiki/API>. Acesso em: Maio de 2006.

Wikipédia; Java (Linguagem de Programação). **Wikipédia a enciclopédia livre**, Junho de 2006. Disponível em: <http://pt.wikipedia.org/wiki/Linguagem_de_programa%C3%A7%C3%A3o_Java>. Acesso em: Maio de 2006.

Wikipédia; Plugin. **Wikipédia a enciclopédia livre**, Maio de 2006. Disponível em: < http://pt.wikipedia.org/wiki/API>. Acesso em: Junho de 2006.

Storkel, S; An a Introduction to the Eclipse IDE. **Site OnJava.com**, Novembro de 2002. Disponível em: <http://www.onjava.com/pub/a/onjava/2002/12/11/eclipse.html>. Acesso em: Junho de 2006.